Lesson 19

The advanced functions of TVPaint Animation

In this lesson, you will learn to :

- Go further about customized panels.
- Use the plug-ins at your disposal.
- Use the « George » scripting language.
- Use the “recorder”.

Customized panels

Adding an action

In the lesson 7, we have seen how to create a custom panel and how to save bryshes and colors inside. However, custom panels may also contain actions such as functions, scripts, etc... Their icons and information may be changed also.

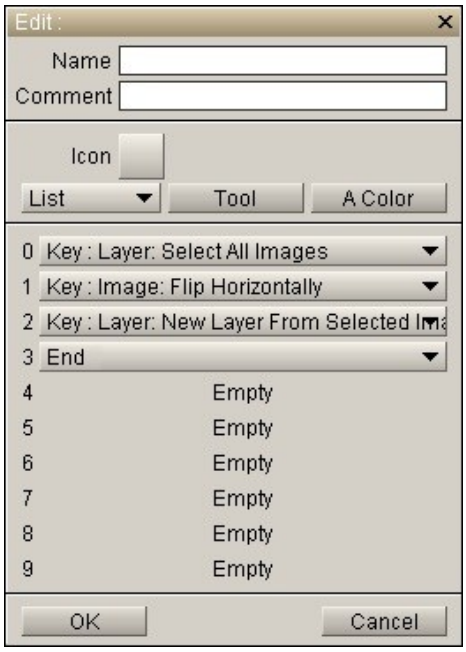
Adding an action is done using the panel at the bottom left of this page :

* Using the first two text fields you may define a name and in-line help comment for your future customized function .
* The buttons and menus *List*, *Tool* and *A color* allow you to choose the icon corresponding to the action you will create.

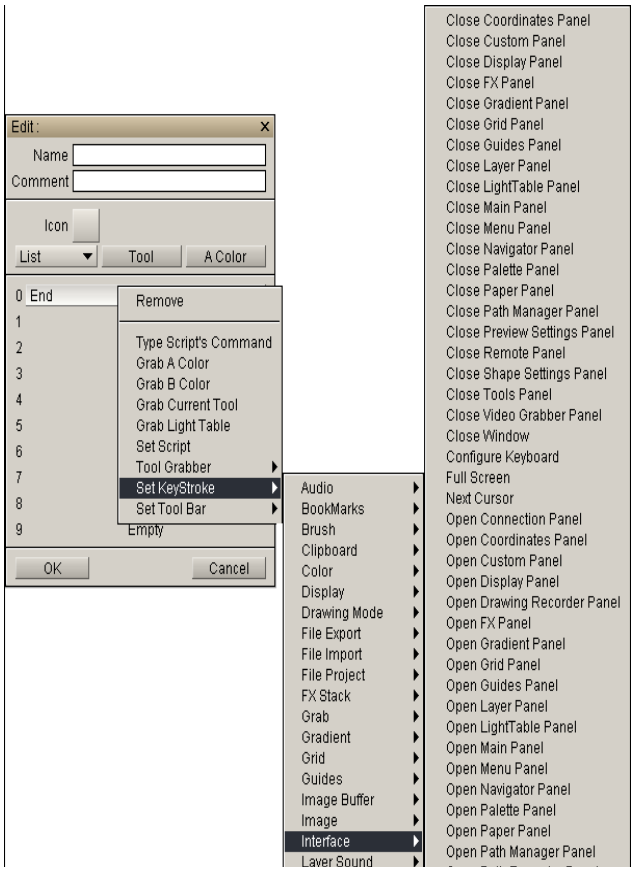
Opposite you find the list of icons supplied with TVPaint Animation.

To create a new icon, you simply have to create a brush with its image and select the *Tool* button.

* The numbered lines are the actions to be carried out one after the other when you click on the icon corresponding to your function.



Above you find the example of an action which selects all images of the current layer, flips them and copies them all to a new layer.



The actions at your disposal.

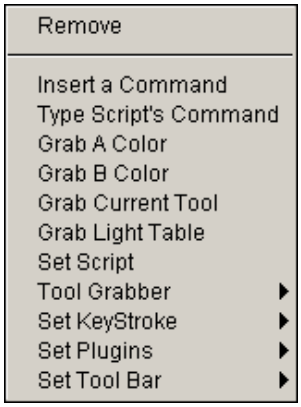
Several possibilities are available if you decide to create an action. You can :

* Select a shortcut in the numerous program functions. (see previous page)

* Select an FX-Bin file stored in the FX-stack to re-use it.

* *Open Plug-in* allows you to place a button to fast-start plugins. (they will be introduced and explained in the next section of this lesson)

* *Open panel* allows you to create dependencies between your panels, i.e. you may display secondary panels from the main panel, for example. This may be useful for organizing your custom brushes or the contents of your various *Bins*.



* Remove the current command.

* Insert a new command.

* Select a script command (for example : TV_Circle, etc ...)

* Grab the *A* color, *B* color or the current tool.

* It is possible to Grab the Light Table.

* You may call up a George automation script.

* Grab the tool only or with the *A* color or with the *B* color or with those two colors.

* Select a KeyStroke, call a plug-in and open a Tool Bar.

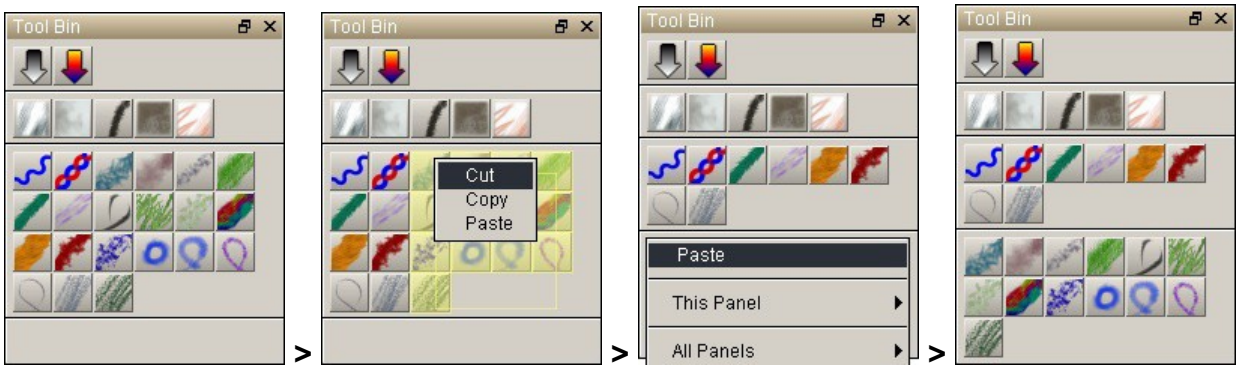
*George Script*s and commands will be discussed further in this lesson.

How to manage the created actions.

Once your buttons are created in the custom panel, other options become available:

* With a simple selection in your custom panel you may cut or copy the icons/actions that you have just created. You simply use the menu described above to « paste » your icons/actions later.

* With a right click on an icon/action in the custom panel, you may edit, duplicate or delete it.



How to use the plug-ins TVP PRO

To use the plug-ins at your disposal, you have to :
* Open a custom panel
* Create a new button in this panel
* Select the plug-in you need as action to do for this button

The panel relative to the plug-in will then be visible once you click on the corresponding button.

The *Canon* Plug-in is available only if you have selected it during the installation process of the software. It is available only on PC computers at this time.

The *Color Factory* plug-in

The *Color Factory* plug-in allows you to modify the values of the R,G,B and Alpha channels of the selected images, thanks to mathematical formulas. (ie : R,G,B and Alpha values of the pixels of your images)

Based on the additive color principle, all colors are mixture of red, green and blue (more or less dark) in exact proportions.

Each color can be broken up in various ways depending on the color system. For example, the *Slider* tab of the *Color Picker* panel gives the numerical values of the A color in the R,G,B and H,S,L color systems.

Each text field of the *Color Factory* panel describes one component (R,G,B or Alpha) and can contain a mathematical formula. The new values of those components will be computed using the formulas you have entered (the Alpha component indicates the opacity of the pixels).

The variables that you can use in your formulas are listed in the table below (all in lower case) :

The variables :	
r	Value of the *Red* component of the pixel
g	Value of the *Green* component of the pixel
b	Value of the *Blue* component of the pixel
a	Value of the *Alpha* component of the pixel
x	x coordinate of the pixel
y	y coordinate of the pixel
w	Width of the current project
h	Height of the current project

And here are listed the arithmetic operators :

Arithmetic Operators		**Examples :**
+	add	18 **+** 3 = 21
–	subtract	18 **–** 6 = 12
*	multiply	18 * 3 = 54
/	divide	18 / 3 = 6

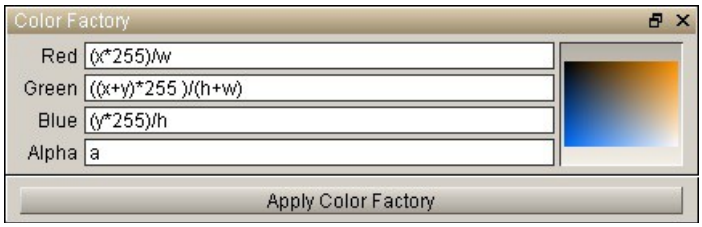
It is possible to use parentheses **( )** in order to set priorities in your computations. If you don't use them, the *Color Factory* plug-in will evaluate the operations from the left to the right.
For instance : 1+2*3 = 9 1+(2*3) = 7

Here are a few examples using the *Color Factory* plug-in.
* Let's begin with a classic image without any modification :

* Here is a negative of the previous landscape :

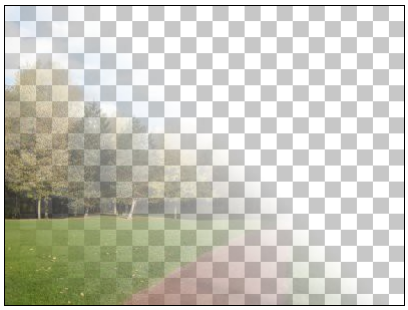
* Below, the red and blue components of the landscape have been inverted : the sky turned into orange and the road turned from red to blue. The green component stay unchanged.

* The *Color Factory* plug-in can also be used to create nice color gradients. Therefore, the formulas on the right panel below are more complicated.



* Below, simple additions have been done with the red and green color channels inverted.

* Here is the last example : an opacity gradient from the top right to the bottom left of the image.



The *Canon* plug-in (PC only)

The *Canon* plug-in allows the user to retrieve photos from a Canon digital camera and to put them directly into a TVPaint Animation layer.
This plug-in is very convenient for the animator who want to work in stop-motion directly with the software.
Here are the steps to use the *Canon* Plug-in with TVPaint Animation :

* What is required :

- The Canon driver of your camera must be correctly installed in your system. (Microsoft Windows only at this time)

- You should have a compatible Canon camera (see the list in Appendix) powered and plugged into an USB port of your computer.

- You have to set your camera in *playback* mode. Don't use the *shooting* mode as it does not allow you to manage your camera via the computer.

Go lesson 5 to know more about the *Video input* panel !

It is advised always to close your Video Grabber and Video Canon panels before quitting the software.

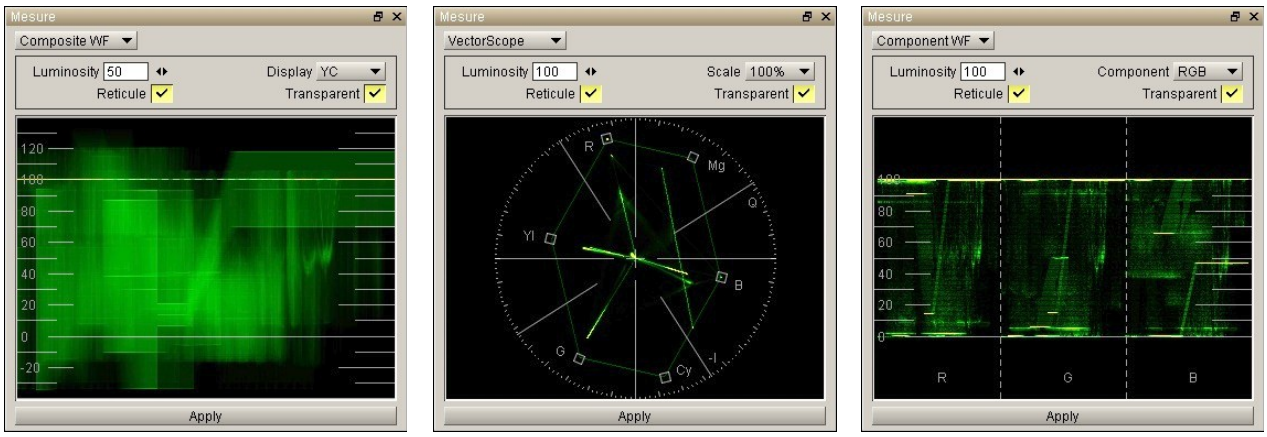
The *Direct Show* plug-in (PC Only)

DirectShow is an application programming interface from Microsoft which contains a set of software libraries. It is a component of DirectX and allows to manage multimedia files.

The DirectShow Plug-in provides support for USB, SCSI, FireWire, ... compatible hardwares.
(in theory all Windows Driver Mode (WDM) compatible hardwares)

The *WaveForm* plug-in

Generally speaking, an oscilloscope is an electrical equipment which allows you to monitor and measure periodic signal voltages. If it is calibrated in order to receive video signals, it is also called WaveForm monitor. In the audiovisual environment, they are often used to check if the transmitted video signal has its characteristics in conformity with the applicable broadcasting rules.



Our WaveForm Plug-in allows the audiovisual professionals to use their classical tools.
Thanks to this Plug-in, it is now easy to check the integrity of a signal, manage calibration and colorimetric adjustments, control the validity of numerical signals, ...

The popup menu allows you either to select :

* A Composite WaveForm monitor
* A VectorScope
* A Component WaveForm monitor

> The monitors at you disposal in this WaveForm Plug-in act as if all layers of the current picture where merged.

Here are some options relative to the different displays at your disposal :

* The Composite WaveForm monitor can use luminance and/or chrominance signal(s).
* The VectorScope can use two calibration sets : 75 % and 100 %
* The Component WaveForm monitor can use the R,G,B or Y,U,V systems.

Let's now explain the other options :

* It is possible to set the *luminosity* of the display thanks to a mini-slider.
* The *reticule* button allows to show or hide the different axis and channels.
* If the transparency button is checked, the black background will not appear when using the *apply* button.
* The *apply* button allows you to copy on the current image the results obtained in the WaveForm Plug-in panel.

How to use the « George » scripting language

Introduction: what is George ?

George is a programming language which allows you to use the numerous tools of TVPaint Animation. Every program written in this language is called "*George Script*". Other softwares use sometimes the word "Macro" instead of "Script".

A *George Script* allows you to execute specific TVPaint Animation command such as :
Create a new layer, draw a circle on this area, save the current project, add a bookmark, etc…

Those *Scripts* are usually created to save some time when executing a repetitive task is needed. Experienced programmers use TVPaint Animation commands in order to create ".dll " plug-ins

Some *George Script*s are available as example with the software, but you can also create your own scripts and/or exchange them with members of the TVPaint Community. (You can access the TVPaint forum here: http://www.tvpaint.com/forum/)

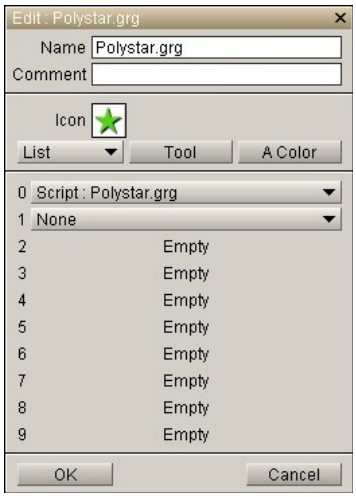
Finding, Editing and Using *George Script*s

After installing TVPaint Animation on your computer, *George Script*s are located in the directory:

C:/program files/TVPaint Animation/George *Windows.*
Applications/TVPaint Animation/Contents/Ressource/George/ *Apple OS-X.*

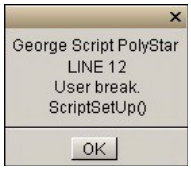
Those *George Script*s are in fact usual ASCII text files and have a ".grg" extension.
It is possible to edit or create them with any text editor like *Notepad* from *Microsoft Windows* or *Text Edit* from Mac OS-X.

The scripts are in theory working on the both standards PC and Macintosh, whatever the computer used to create them.



To use one of the *George Script*s at your disposal, just follow the steps below:
_ open a custom panel
_ create a new button on this panel
_ select the action: -*Set script*- for this button
_ select a file with a ".grg" extension in the file requester

Once finished, if you click on the just created button, the *George Script* will then be launched.



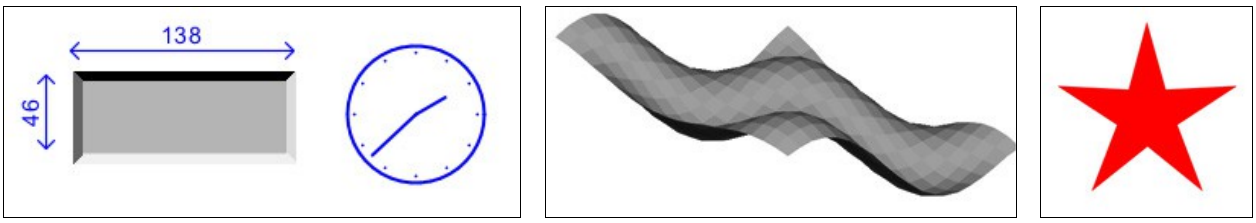
At every moment, you can stop the execution of a *George Script* by using the [Esc] key.

A window, indicating on which line the execution of the script was stopped, should popup.

Once a George Script has finished its programmed tasks, the Undo option of the TVPaint Animation main panel allows you to cancel all the changes it has generated.

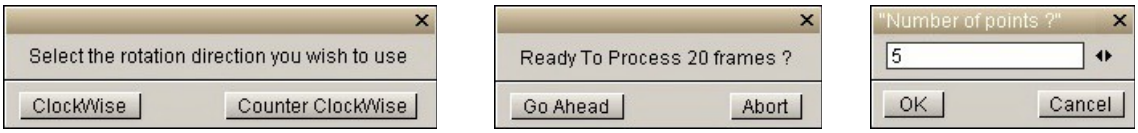
A few examples of *George Script*s

Often, *George Script*s need an action from the user in order to be launched. For instance :
The *George Script* named "*long.grg*" waits for a segment to be drawn on screen and finally returns its length.
The *George Script* called "*clock.grg*" waits for a circle to be drawn on screen and finally draws a clock inside this one.
The *George Script* called "*button Z.grg*" waits for a rectangle to be drawn on screen and returns a bevelled button inside it
A few *George Script*s don't need any action from the user to be started: it's the case with the *George Script* called "*Zsurface.grg*".



Above: the results of the scripts: button Z, Long, Clock, Zsurface and Polystar

A few *George Script* are more complicated and some requesters may appear.
It is the case with the *George Script* called *Polystar.grg* : during its execution, a first requester asks you to specify how many vertices should be drawn. If you apply your script on an animated layer (after selecting all its frames), a second and third requesters will ask you more informations (see below)



Programming in George language

George is a scripting language which can do the basics:
- Loops (do the same thing n times),
- Conditional testing (if this is true then do that),
- Computations (a+b, sin(a),...).

The George language is not particularly fast, but it is flexible and very easy to use. On many points, George is comparable to the Pascal language or other well known programming language. However it's not aimed at developing heavyweight applications as are for example the C or C++ languages. George is first and foremost a scripting language.

Instructions and Commands

George needs classical instruction to structure the scripts: tests, loops, computation, conditional events, etc… For a better readability, all the instructions will be written in green in this manual.
For example :

If, Else, End, For, Do, Until, While, …	for conditionals tests
Cos, Sin, Tan, Rnd, …	for computations

You can also call in your scripts TVPaint Animation commands, which call specific tools of the software.

They can be differentiated from the classical George instruction because they all begin with the « **tv_** » prefix. (They will be written in blue color)
For instance :

* The command **tv_LayerCreate** allows you to create a new layer.
* The command **tv_SaveBrushAnim** saves the current brush on the hard drive.
* It is possible to change the colors of the bin tab thanks to the **tv_SetPalette** command
* Etc… (You will find in appendix the complete instruction and command listings of the George Language.)

All TVPaint Animation commands need their own arguments and return some values as answer.
For example, the tv_LayerCreate command needs a name for the layer that you want to create.

This command returns some information regarding your layer: opacity, number of pictures, place of the timeline,… So you can reuse or modify them.

It would be too long to enumerate here the arguments and the return values of each George command. All those parameters are fully explained in the developper kit (SDK) of TVPaint Animation. The SDK is available on demand at the email address: tvpaint@tvpaint.fr

Make sure to differentiate George instructions (which often start with a sharp : **#**) and George commands for TVP Animation (which always start with the « **tv _** » prefix).
The commands act on the software itself and on your current display whereas George instructions are used to write the structure of the program (conditionals, computations, loops, ...)

Since approximatively 10 years, all the softwares using the TVPaint technology offers a scripting language.
Here are some informations regarding the compatibility between those scripting language : If you upgraded to TVP Animation, your old scripts should still work.
However, some recent commands (for instance: **TV_AddBookmarks**) are not available in older products using the TVPaint Technology. (Aura, Mirage,…)

An example of program written in George.

The program on the next page comes from the file clock.grg. It allows you to draw a clock displaying the current time on the project window.

We suggest you create a button assigned to this script in a custom panel, as explained previously. After this operation, it should be easy for you to test this *George Script* in the project window.

Once this stage is passed, you can try to edit the *George Script*, to modify it and to test it again in TVPaint Animation. This should help you learn the basics of the George Language.

The best way to get used to George scripts is probably to try the example programs that come with TVPaint Animation.

Study them and make small alterations to them, then run them and see the effects of your changes.

The *clock.grg* script :

```
Param Circle
                // TVPaint should wait for the coordinates of a circle before running the
                // program. The user must therefore draw a circle to start the program.

Parse result command x y r button
                // Fetches the coordinates of the user's circle.

if cmp(command,"Circle")==0
        tv_Warn "This program needs the coordinates of a circle !"
        exit
                // If there are no coordinates for a circle, warn the user and end the program.
End


t=time
parse t h m s
                // Fetches the hours (h), minutes (m) and seconds (s) of the current time.
m=m+s/60
                // The seconds can slightly alter the minutes arm of the clock.
tv_Pen r/30             // Selects a pen proportional to the clock size.
tv_Circle x y r                 // Draws the outline of the clock.
ang=360/12
d=r-(r/10)
#for i=0 to 360-ang step ang
                        // Loop to draw the 12 hour markers on the clock
        #a=cos(i)*d
        #b=sin(i)*d
        tv_Dot x+a y-b 0
#end


i=h*360/12              // Draw in the hours
d=r/2
a=sin (i)*d
b=cos (i)*d
tv_Line x y x+a y-b


i=m*360/60              // Draw in the minutes
d=r-(r/8)
a=sin(i)*d
b=cos(i)*d
tv_Line x y x+a y-b
                        // End of the example
```

Commands Description

Here are some information regarding the commands that you can find on the previous page :

Below, each command will be followed by the syntax of the variables it requires.
For example, to draw a straight line from one point to another, you must give the coordinates of the start point then the end point.

The syntax to draw such a line would be:

tv_Line x1 y1 x2 y2 [0/1]

The **[0/1]** parameter indicates that you can use either the left **[0]** or the right **[1]** mouse button. This is optional; by default, it's always the left button that will be used.

To summarize :

[ ] Variables inside square brackets are optional.
/ The "/" symbol separates several options of which only one is used.
For instance, 0/1 means either 0 or 1.

tv_Circle x y r [0/1]
This command allows you to draw a circle with a radius r and a center located on the pixel with (x,y) coordinates. The **[0/1]** parameter indicates that you can use either the left **[0]** or the right **[1]** mouse button.
Of course, the circle will be drawn on the current image on the current layer.

tv_Dot x y [0/1]
this command allows you to use the current tool (airbrush, pen, custombrush,…) on the pixel with (x,y) coordinates, in the current image on the current layer.
The **[0/1]** parameter indicates that you can use either the left **[0]** or the right **[1]** mouse button.

tv_Warn text
This command displays a popup dialog box containing the character string **text** and an "ok" button. The character string must be surrounded by quotation marks.

tv_Pen size
This command allows you to select the pen tool with its current parameters (drawing mode, opacity, power, ...) and change its size.

You should always remember that a George Script will use TVP Animation as it finds it, i.e. with the functions and options that are currently set.

For instance, if you want to draw a red line with a George Script, the script needs to be told more than just the line coordinates. Unless you explicitly tell it to use a red color, your script will go ahead and use the current color when the script is run, whatever it may be.

Similarly, if when you run the script it is in the *Behind* or *Colorize* drawing mode, you won't exactly be getting the clean line you were hoping for.

A few useful options

It is now time to explain some programming conventions:

1°) A sharp character # at the beginning of a line will not affect the execution of the script.
For instance, the line :

```
Pause 20
```

will pause the execution of your script during 20 seconds.
And the following script line will have the same result :

```
#Pause 20
```

2°) You can use only one instruction or command on the same line. For instance, if you use the following line in a script :

```
Print "wait a little" Pause 20
```

An error message will be returned.
The two lines below will not return any error messages :

```
Print "wait a little"
Pause 20
```

3°) Everything on a line that is after a double slash // is ignored. They are only comments that have been put into the script to make it more understandable and readable by humans. For instance, the line :

```
Pause 20 // wait 20 seconds
```

makes TVPaint Animation wait during 20 seconds.
However, the line :

```
// Pause 20
```

will not pause the execution of your script.

TVP Animation sees no difference between uppercase and lowercase letters. For instance, **TV_layercreate, tV_LaYeRcReAtE, tv_LAYERCREATE** are all exactly the same command.

How to use the variables

It is not necessary to declare the types of your variables in a George Script.
For instance, you can write the two lines below in a script without declaring that the variable *MyNumber* is numeric or the variable *Answer* boolean.

```
MyNumber=10
Answer=false
```

As for commands and instructions, the George language sees no difference between uppercase and lowercase letters. For instance, *MyNumber, mynumber, MYNUMBER, MyNuMbEr* are all exactly the same variable.

The George language use a few reserved variables in order to work correctly. You can use them as you wish, but it is not advised to modify them.

That is the case with the two variables « *Time* » and « *Result* » (encountered in the clock.grg script above) which contains respectively the current time and the results of the last used TVP Animation command.

The operators at your disposal

Arithmetic Operators		Examples :
-	negative	18 -> -18
**	exponent	18**3=18*18*18=5832
*	multiplication	18*3=54
/	division	18/3=6
+	addition	18+3=21
-	subtraction	18–6=12

Logic Operators		Examples :
&&	and	a&&b
II	or	aIIb

Related Operators		Examples :	
==	equal	x==6	x==x+2
!=	not equal to	x!=5	x!=y-3
>	greater than	x>4	x>y-z
>=	greater or equal than	x>=3	x>=z/y
<	less than	x<8	x<-y
<=	less or equal than	x<=7	x<=x+5

The « = » operator is an assignment operator and allows to assign a given value to the variable of your choice. For example :

```
MyNumberOfLayers=5
```

The « == » operator is an comparison operator between two variables, usually used during conditional tests. For example :

```
if a==5
```

The comparison operator is not appropriate if want to know if two non numerical variables are equal. For this, you need to use the instruction **cmp**, as shown in the *clock.grg* script above :

```
if cmp(command,"Circle")==0
```

The Param and Parse instructions, the launching modes.

The instruction **Parse** allows you to split up a character string into smaller character strings

```
Rainbow="red orange yellow green blue indigo violet"
Parse  Rainbow color1 color2 color3
tv_warn  color1        //  displays « red »
tv_warn  color2        //  displays « orange »
tv_warn  color3        //  displays « yellow green blue indigo violet »
```

Usually, each *George Script* begins with the **Param** instruction. It allows you to choose which action will have to be done by the user in order to start the script execution : drawing a circle, a segment, a rectangle ...

* The script line :

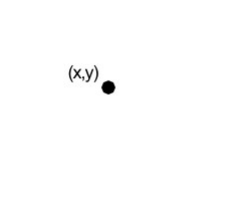
```
Param None
```

allows the *George Script* to be executed without waiting for an user action.

* The two script lines :

```
Param Single
Parse result command x y button
```

allows you to execute a *George Script* after a simple mouse click on the project window.

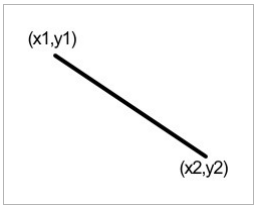


- The character string *« single»* will be assigned to the variable called *command* if a point is drawn on the screen.
- The point coordinates will be assigned to the variables x and y.
- The button selected by the user to draw the point will be assigned to the variable called *button*. (1 for RMB, 0 for LMB).

* With the following script lines :

```
Param Line
Parse result command x1 y1 x2 y2 button
```

The *George Script* will be launched only if a segment is drawn on the project window.

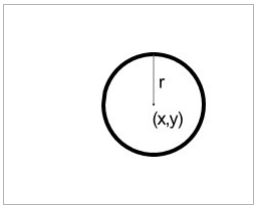


- The character string *« Line »* will be assigned to the variable called *command* if a line is drawn on the screen.
- The coordinates of the points at the ends of the segment will be assigned to the variables x1 y1 and x2 y2 (see opposite)
- The button selected by the user to draw the line will be assigned to the variable called *button*. (1 for RMB, 0 for LMB).

* With the two script lines below :

```
Param Circle
Parse result command x y r button
```

The *George Script* will be launched only if a circle is drawn on the project window.

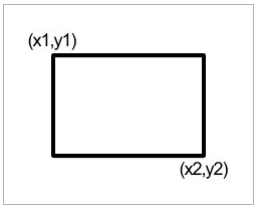


- The character string *« Circle »* will be assigned to the variable called *command* if a circle is drawn on the screen.
- The coordinates of the circle center will be assigned to the variables x and y.
- The radius of the circle will be assigned to the variable r.
- The button selected by the user to draw the circle will be assigned to the variable called *button*. (1 for RMB, 0 for LMB).

* With the following script lines :

```
Param Rectangle
Parse result command x1 y1 x2 y2 button
```

The *George Script* will be launched only if a rectangle is drawn on the project window.

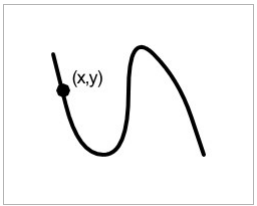


- The character string *« Rectangle »* will be assigned to the variable called *command* if a rectangle is drawn on the screen.
- The coordinates of the points at the ends of the rectangle will be assigned to the variables x1 y1 and x2 y2 (see opposite)
- The button selected by the user to draw the rectangle will be assigned to the variable called *button*. (1 for RMB, 0 for LMB).

* With the two script lines below :

```
Param Freehand
Parse result command x y button
```

The *George Script* will be launched only if a freehand curve is drawn on the project window.



- The character string *« Freehand »* will be assigned to the variable called *command* if a curve is drawn on the screen.
- The button selected by the user to draw the curve will be assigned to the variable called *button*. (1 for RMB, 0 for LMB).

The character strings

It is important to know how to use the character strings because they usually plays a part in numerous situations.
For example, in the following script lines, a loop allows to load the projects :
D:/cartoon3.tvp, D:/cartoon4.tvp, ... D:/cartoon15.tvp, D:/cartoon16.tvp.

```
For number=3 to 16
        NameOfTheFile="D:/cartoon"number'.tvp'
        tv_loadproject NameOfTheFile
End
```

As shown above, you will have to use simple quotation marks ('...') or double quotation marks ("...") in order to differentiate variables from character strings.

In the example above :

number and ***NameOfTheFile*** are variables
D:/cartoon and ***.tvp*** are character strings

The variable ***NameOfTheFile*** receive the concatenation of ***D:/cartoon, number*** and ***.tvp***
In other words, the character strings ***d:/cartoon*** and ***.tvp***, the variable ***number*** have been joined end to end in order to create the variable ***NameOfTheFile.***

It is possible to use the **Concat** instruction to join end to end character strings and/or variables. The following lines have the same effect :

```
NameOfTheFile=number'.jpg'
NameOfTheFile=Concat(number,.jpg)
```

If you need to use simple quotation marks ('...') or double quotation marks ("...") in your character strings, you must surround them with the other quotation marks at your disposal, as shown in the example below :

```
Print  ' My prefered animation program is " TVP Animation " '
                    // will surround the name of the software with double quotation marks.
Print  " My prefered animation program is ' TVP Animation ' "
                    // will surround the name of the software with single quotation marks.
```

The arrays

As in other programming languages, George can create and use arrays with one, two or more dimensions.

To create an array, you only need to add parentheses or square brackets after the name of a variable. You can use either parentheses or square brackets.

For instance,
MyArray(i) or MyArray[i] represent the same array with one dimension.
MyArray(i,j) or MyArray[i,j] represent the same array with two dimensions.
MyArray(i,j,k) or MyArray[i,j,k] represent the same array with three dimensions.
And so on ...

With the George language, it is not necessary to indicate or declare the dimension or the number of elements before using an array.

In the exemple below, an array called *color* with one dimension was created.
It contains four elements : "Magenta", "Cyan", "Yellow" and "Black"
We used a loop to display its content.

```
color[1]="Magenta"
color[2]="Cyan"
color[3]="Yellow"
color[4]="Black"

For i=1 To  4

     tv_warn color[i]          //  displays each color : "Magenta", "Cyan", "Yellow"
                               // and  "Black", in a dialog box.
End
```

Functions and procedures

Let's assume that you are writing a script which needs some instructions, conditional tests, etc ... to be executed several times. (For instance, a script which tests the pixel's colors several time and acts differently depending on the obtained results.)

Copying the same lines several time in a George Script can quickly become fastidious. It doesn't make the things easier when you need to check or modify your script.
That's the reason why writing procedures and functions is very convenient :

* A function is a set of instructions and commands which allows you to execute a precise task inside a George Script. The functions are independent from the remainder of the George Script and are located at the end of the script.

The part of a George Script which distinguish itself from the function(s) will be called “main part of the script”. All the functions are available at any moment in order to be used by the Main part of the script. A function can receive parameters and always returns a result.

* A procedure is a function which does not return results.

Here is an example of a script using a function :
(Of course, using functions is really useful when you have to create bigger and more sophisticated scripts)

```
Param Single
Parse result command abscissa ordinate button
tv_warn TestBlackColor(abscissa,ordinate)

//  Main part of the script ▲
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//  Function ▼

Function TestBlackColor(x,y)
        Local r g b alpha sum
        tv_getpixel x y
        Parse result r g b alpha
        If alpha<>0
                sum=r+g+b
                If  sum==0
                        Return " the pixel's color is black "
                Else
                        Return " the pixel's color is NOT black "
                End
        Else
                Return " the pixel is transparent "
        End
End
```

The line :

tv_warn TestBlackColor(abscissa,ordinate)

Can be replaced by the two following lines :

```
Answer=TestBlackColor(abscissa,ordinate)
tv_warn Answer
```

Here is the syntax of the instructions relative to the procedures and functions :

Function NameOfTheFunction (Variable1, Variable2, Variable3, etc ...)
This instruction declares the existence of a function or a procedure in a George Script. (there is no instruction called « procedure »)
This instruction is followed by the procedure or function name, and then by the name of the variables (from the main part) that are needed.

Functions and procedures must always be located after the main part of your script. Calling a function or a procedure in the main part of your script can be done by writing its name and the needed variables between parenthesis, separated by comas.

Local LocalVariable1, LocalVariable2, LocalVariable3, etc ...
This instruction allows you to specify the variables which will be used solely inside your function or procedure.

Return Result
This instruction allows you to specify the result of your function and to make it available in the main part of the script. It could be a character string, a numerical value, a boolean value, etc ...
In the main part of your script, it is possible to save it into a variable, and after to re-use it.
This instruction is optional. If you don't use it, you have created a procedure instead of a function.

End
This instruction is needed to finish your function or procedure.

The functions and procedures written in George language can be recursive. In other words, a function can call itself. It allows you to create more sophisticated scripts.

For instance, drawing a fractal curve can be done thanks to a recursive script (see below)

How to create libraries of functions and procedures.

We have just seen in the last chapter that it was possible to create your own George functions and procedures.
This may be taken even further as you can store your functions and procedures in «.grg» files and you can re-use them in other scripts written in George.
In other words, you can create your own libraries.

To insert one of your libraries in a George Script, you only have to write the following line in your script :

```
#Include "MyLibrary.grg"
```

The instruction ***#Include*** will search the file called *MyLibrary.grg* in the directory containing the current George Script or in the root directory of TVPaint Animation.

The following line allows you to specify a more precise access path, from the base directory :

```
#Include "MyFolderOfLibraries/MyLibrary.grg"
```

Please read the ***Initialization File*** section in the appendix of this user-manual if you want to change the root directory of TVP Animation.

Two examples of libraries are available in the directory :

C:/program files/TVPaint Animation/George/Include *Windows*
Applications/TVPaint Animation/Contents/Ressource/George/ *Apple OS-X*

They are called *Basic.grg* and *Advanced.grg* and are related to character strings management.

```
# Include "include/basic.grg"
# Include "include/advanced.grg"
```

To make them available in the George Scripts given with the software, you only need to add the opposite lines at the end of those scripts :

The *Startup.grg* file

At startup, the first thing that TVPaint Animation does (after displaying the main toolbox, but before giving control to the user) is to run the *Startup.grg* script (which should be in the same directory as the other scripts).
When there is no *Startup.grg* file, nothing will happen. Otherwise, you have at your disposal a startup file that gets run every time you load TVPaint Animation.
If, for example, you rename the *Logo.grg* file to *Startup.grg*, it will load the TVPaint Développement logo to screen every time you run TVPaint Animation.
You could also, thanks to a simple George script, define all your preferred parameters. For instance : load a project from your harddrive, change the current colors, add some layers to your project, etc…

The TVPaint Animation Plug-in developpement

As explained before, George is not the only program that can send commands to TVPaint Animation.
Any other program could theoretically do so. Examples using an Excel script, a C program or other languages are certainly conceivable.

Besides, it is possible for experienced users or third-party companies to create *Plug-ins* that can be used with TVPaint Animation.
Those *Plug-ins* allows you to add customized tools to the software like effects on pictures, new drawing modes, new tools or new functions.
The Plug-in may be in the « .dll » (*Dynamic Link Library)* file format, which can be obtained thanks to a compiler. « .dll » files are not readable in a text editor, unlike the classical George Scripts.

For more details about this subject, please visit the TVPaint Developpement forum. You can also ask us the development kit (also called SDK) at the email address : tvpaint@tvpaint.fr
This kit contains source codes for interface functions, documentation and some detailed plug-in examples.

Adding arguments

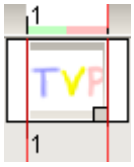
You can add arguments to the execution of TVPaint Animation. These arguments are :
- images (just write the path and file name),
- scripts (syntax: script=PathAndFileName.grg),
- george commands (syntax: cmd=GeorgeCommand).

For example :
TVPaint Animation.exe "MyImage" "script=MyScriptPathAndName.grg"
"cmd=MyGeorgeCommand"

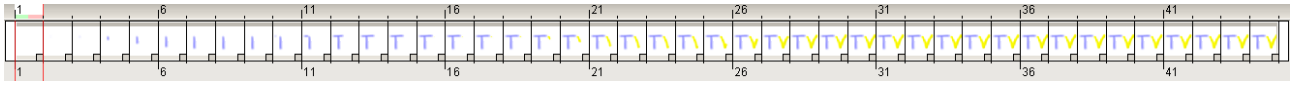
The Drawing Recorder

The aim of the *Drawing Recorder* panel (*Windows* > *Utilities*) is to store all the strokes of the user when he is drawing. With this feature, it is possible to create an animation from a static drawing. So the result of the *Drawing Recorder* is an animation which shows each step of the drawing and how it has been constructed. Note that it is possible to change the current tool during the drawing and the timing of the drawing is preserved when the animation is created.

Each time the content of the image changes, a new image is added in the animation of the *Drawing Recorder*.



The original image which was drawn, while the *Drawing Recorder* was running, 3 letters with 3 different tools.



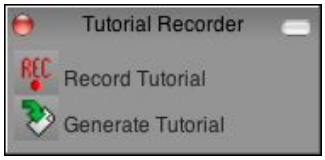
The animation, based on the original image, created thanks to the drawing recorder.

The *Drawing Recorder* panel is very simple, three buttons and a preview.

* The menu above allows you to choose to preserve the alpha channel (*Current layer*) or not (*Display*).
* The preview area and the slider below it allow to see and navigate between each frame of the record.
* The Record/Pause button starts or pauses the recording.
* When the user thinks he has finished his work, the new Project button allows you to create a new project and store all the strokes/frames of the recording.
* And with the Skull button , it is possible to delete the current recording to be able to start a new one.

The *Tutorial recorder*

This custom panel allows you to record easily your TVPaint Animation interface. This is really useful, especially to create tuorials. This panel can be found in the menu *Custom panelsî.*



The first button is used to start the record, whereas the second allows to stop and generate an images sequence into the TVPaint Animation's timeline.